

# A Base for Achieving Semantics for Prolog with Cut for “Correct” Observables

Lingzhong Zhao<sup>1,2</sup>, Tianlong Gu<sup>2</sup>, Junyan Qian<sup>2</sup>, and Guoyong Cai<sup>2</sup>

<sup>1</sup>Electronic Engineering School, Xidian University, Xi'an, China 710071

<sup>2</sup>Department of Computer Science, Guilin University of Electronic Technology, Guilin, China 541004  
gllzzhao@gliet.edu.cn

## Abstract

*Decorated tree semantics is a goal-independent denotational semantics for Prolog that deals with the control rules of Prolog and cut operator. This semantics was designed with the aim to provide a general framework for abstract analysis of generic properties of logic programs and has been specialized into computed answer (ca) semantics and call pattern (cp) semantics. In this paper we show that the methods for achieving semantics for the “correct” observables (i.e. observables related to successful derivations), correct partial answers (cpa) and correct call patterns (ccp) in particular, are not trivial extensions of the methods for achieving ca and cp semantics. We extend the work in [13] and propose a two-step method for achieving cpa and ccp semantics using the theory of abstract interpretation. This paper is concerned with the first step, where the main result is a modified decorated tree semantics that serves as a base for further abstraction to get cpa and ccp semantics.*

## 1. Introduction

The development of appropriate semantics for Prolog is the base for semantics-based Prolog program verification and analysis, which are important in developing correct and efficient Prolog programs. This work can be viewed as a continuation of the work in [13], where a goal-independent denotational semantics, i.e. decorated tree (DT) semantics, was proposed by Spoto. This semantics is a further development of a series of semantics for Prolog and is suitable for goal-independent program analysis [2,11,12]. A feature of this semantics is that it uses syntactic objects (substitutions with some control decoration) rather than functions as the denotational domain, which makes it suitable for abstract interpretation. This choice coincides with the idea of s-semantics [3].

Just as the semantics presented by Charlier et al. [9,10], Spoto’s semantics was designed with the aim to provide a general framework for the abstract analysis of generic properties of logic programs. As stated in [13], DT semantics “is a very concrete denotational semantics that can be specialized into a more abstract one as soon as we specify the observable property of interest.” According to

[6], a large class of observable properties of Prolog programs can be captured by different kinds of observables, which include computed answers, (correct) partial answers, (correct) call patterns, successful derivations and finite failures etc. Semantics for computed answers and call patterns have been given as specializations of DT semantics and further abstractions of those two semantics have been proposed to get finitely computable abstract semantics for program analysis [14].

Our interest in this paper is related to the question whether the semantics for other observables can be effectively obtained as abstractions of DT semantics. We argue that the task of achieving semantics for “correct” observables, cpa and ccp in particular, from DT semantics is not trivial. By *correct* observables we mean those related to successful derivations. Our work shows that cpa (or ccp) semantics can be achieved systematically by first abstracting DT semantics to an intermediate modified decorated tree (MDT) semantics and then achieving cpa (or ccp) semantics as an abstraction of MDT semantics. This paper is concerned with the first step.

## 2. Preliminaries

We assume the familiarity with the basic algebraic structures and Prolog language. The basic notations are used in the usual way. A sequence is an ordered collection of elements possibly with repetitions. The set of all possible sequences of elements of set  $E$  is denoted by  $\text{Seq}(E)$  and the set of non-empty sequences of elements of  $E$  is denoted by  $\text{Seq}^+(E)$ .  $\cdot$  represents sequence concatenation. A sequence is denoted by a variable with a tilde sign on it. The empty sequence is explicitly written as  $\epsilon$ .  $\# \tilde{s}$  is used to denote the length of the sequence  $\tilde{s}$ .

Abstract interpretation is a general theory of semantics approximation. In this paper it is used for relating formal semantics at different level of abstractions [4,5]. A common method for formalizing abstract interpretation is by means of a Galois connection.

**Definition 2.1** Given two partial ordered sets (posets)  $\langle P, \sqsubseteq \rangle$  and  $\langle P^a, \sqsubseteq^a \rangle$ , a Galois connection between them is a pair  $\langle \alpha, \gamma \rangle$  such that  $\alpha$  is a total map from  $P$  to  $P^a$ ,  $\gamma$  is a total map from  $P^a$  to  $P$ , and  $\forall p \in P: \forall p^a \in P^a: \alpha(p) \sqsubseteq^a p^a \Leftrightarrow$

$p \sqsubseteq \gamma(p^a)$ .  $\alpha$  and  $\gamma$  are respectively called the abstraction and the concretization maps of the connection.

The abstract interpretation using Galois connection can be formalized as follows.

**Proposition 2.2** (Proposition 23 in [4]) Let  $\langle P, \sqsubseteq \rangle$  and  $\langle P^a, \sqsubseteq^a \rangle$  be two posets with  $\perp$  and  $\perp^a$  as the minimal elements, respectively,  $\langle \alpha, \gamma \rangle$  is a Galois connection between  $P$  and  $P^a$ ,  $\perp^a = \alpha(\perp)$ ,  $\varphi: P \mapsto P$  is such that  $\text{Lfp}(\varphi) = \text{lub}_{n \geq 0} \varphi^n(\perp)$  (where  $\text{lub}$  is the least upper bound operator) and  $\varphi^a: P^a \mapsto P^a$  is monotonic, then  $\alpha \circ \varphi = \varphi^a \circ \alpha$  implies  $\alpha(\text{Lfp}(\varphi)) = \text{Lfp}(\varphi^a)$ . The condition  $\alpha \circ \varphi = \varphi^a \circ \alpha$  is called local correctness condition and  $\alpha(\text{Lfp}(\varphi)) = \text{Lfp}(\varphi^a)$  is called global correctness condition.

By Tarski's fixpoint theorem [15] the following proposition obviously holds as a consequence of proposition 2.2.

**Proposition 2.3** Let  $\langle P, \sqsubseteq, \perp \rangle$  and  $\langle P^a, \sqsubseteq^a, \perp^a \rangle$  be two complete lattices with  $\perp$  and  $\perp^a$  as the minimal elements, respectively,  $\langle \alpha, \gamma \rangle$  is a Galois connection between  $P$  and  $P^a$ ,  $\varphi: P \mapsto P$  and  $\varphi^a: P^a \mapsto P^a$  are two operators such that  $\varphi$  is continuous and  $\varphi^a$  is monotonic, then  $\alpha \circ \varphi = \varphi^a \circ \alpha$  implies  $\alpha(\text{Lfp}(\varphi)) = \text{Lfp}(\varphi^a)$ .

In this paper proposition 2.3 is used to prove the correctness of abstract semantics. Moreover, the proof can be further simplified using the following result.

**Proposition 2.4** Let  $\langle P, \sqsubseteq \rangle$  and  $\langle P^a, \sqsubseteq^a \rangle$  be two posets with  $\perp$  and  $\perp^a$  as the minimal elements, respectively, and  $\varphi: P \mapsto P$ ,  $\varphi^a: P^a \mapsto P^a$  and  $\alpha: P \mapsto P^a$  be three monotonic maps such that  $\alpha(\perp) = \perp^a$  and  $\alpha \circ \varphi = \varphi^a \circ \alpha$ . Then  $P$  and  $P^a$  can be extended to two complete lattice  $P'$  and  $P'^a$ , respectively, and  $\alpha$ ,  $\varphi$  and  $\varphi^a$  to continuous maps  $\alpha'$ ,  $\varphi'$  and  $\varphi'^a$  such that  $\langle \alpha', \gamma' \rangle$ , for a suitable  $\gamma'$ , is a Galois connection between  $P'$  and  $P'^a$ ,  $\alpha'(\perp) = \perp^a$  and the correctness condition  $\alpha' \circ \varphi' = \varphi'^a \circ \alpha'$  holds.

By the above proposition we only have to define semantic operators and abstraction map on posets instead of complete lattices when showing the correctness of abstract semantics.

An abstract syntax was used for Prolog programs in Ref. [13] to simplify the semantic operators. The basic idea is to look at Prolog as an instance of the general CLP scheme [8]. Without loss of generality all the predicates in Prolog programs are assumed to be unary. The clause has the form  $p(x) :- G_1 \text{ or } \dots \text{ or } G_n$  with  $n \geq 1$ , where  $G_1, \dots, G_n$  are goals defined by the grammar:

$G ::= c | p(x) | G$  and  $G | \text{exists } x. G | \text{cut}(G) | \text{cut}_d(G)$ , where  $c$  is a constraint,  $x$  is a program variable,  $d$  in construct  $\text{cut}_d(G)$  is a non-negative integer which represents the effect scope of a cut operator. The cut

operator in  $\text{cut}(G)$  is assumed to have an infinite effect scope and is called open cut, and that in  $\text{cut}_d(G)$  is called closed cut. The set of all possible goals is denoted by  $\mathbb{G}$ . The expression  $p(x)$ , where  $p$  is a predicate symbol, is called *procedure call*.

The constraints domain from which  $c$  is taken is defined as a lattice  $\langle \mathcal{B}, \leq, \vee, \wedge, \text{true}, \text{false} \rangle$  [7,13]. It is assumed that  $\mathcal{B}$  contains the element  $\delta_{x,z}$  for each pair of variables  $x$  and  $z$ , which represents the constraint identifying the variables  $x$  and  $z$ . Moreover, there is a family of monotonic operators  $\exists_x$  on the set of constraints, representing the restriction of a constraint obtained by hiding all the information related to the variable  $x$ .

A goal is called *divergent* if and only if it contains a procedure call. The denotation  $\text{div}(G)$  means  $G$  is divergent. A goal which is not divergent is said to be *convergent*. The function  $\text{con}(G)$  associates to  $G$  the conjunction of the constraints that precede the first procedure call in  $G$ . Function  $\text{con}(G)$  is defined as follows.

$$\text{con}(c) = c, \quad \text{con}(p(x)) = \text{true}$$

$$\text{con}(\text{exists } x.G) = \exists_x.\text{con}(G)$$

$$\text{con}(\text{cut}(G)) = \text{con}(\text{cut}_d(G)) = \text{con}(G)$$

$$\text{con}(G_1 \text{ and } G_2) = \begin{cases} \text{con}(G_1) & \text{if } \text{div}(G_1) \\ \text{con}(G_1) \wedge \text{con}(G_2) & \text{if not } \text{div}(G_1). \end{cases}$$

### 3. Decorated Tree (DT) semantics

Spoto's DT semantics follows a goal independent approach, which means that the behavior of a goal in a program is defined as the evaluation of the goal in the denotation of the program. The DT semantics also features its way of dealing with Prolog control and cut operator. Roughly, the program denotation associates to any predicate in a Prolog program a decorated tree that describes all the possible SLD-derivations from the predicate. An observability constraint in a node of the tree tells how the node is affected by divergent computation or by cut operators, and therefore it can be used to determine whether a Prolog interpreter can actually visit the node. This is the main idea of the so-called technique of "control compilation" [1,13].

The concept of tree is basic in the Spoto's DT semantics. A tree is an element of the set

$$\mathbb{T} = \{ (G, \varepsilon) \mid G \in \mathbb{G} \}$$

$$\cup \{ (G, \tilde{t}) \mid G \in \mathbb{G}, \text{div}(G) \text{ and } \tilde{t} \in \text{Seq}(\mathbb{T}) \}$$

A tree is called *i-cut-closed* if and only if every  $\text{cut}_d(G)$  construct is at the height at least  $(d-i)$  from the root. It is assumed that the root has height 0 and its children have height 1, and so on. An *i-cut-closed* tree is such that, when attached to the leaves of another tree, its cuts can

only affect the last  $i$  levels of nodes of that tree. A tree is *cut-closed* when it is 0-cut-closed.

A decorated tree is achieved by adding an observable constraint part to the nodes of a tree. An observability constraint can be seen as a set of basic constraints of  $\mathcal{B}$ . The set  $\mathcal{O}$  of observable constraints is formalized in definition 3.1.

**Definition 3.1** The set  $\mathcal{O}$  of observability constraints is defined as the minimal set containing  $\mathcal{B}$  and such that if  $S \subseteq \mathcal{O}$  then  $\sqcap S$  and  $\sqcup S$  belong to  $\mathcal{O}$  and such that if  $o \in \mathcal{O}$  then  $\neg o \in \mathcal{O}$ . We will often write  $o_1 \sqcap \dots \sqcap o_n$  for  $\sqcap \{o_1, \dots, o_n\}$  and similarly for  $\sqcup$ . Moreover  $true_o$  and  $false_o$  are shorthand for  $\sqcap \{o\}$  and  $\sqcup \{\neg o\}$ , respectively. An observability constraint  $o$  is true if and only if  $o \in \mathcal{B}$  and  $o \neq false$ , or  $o = \sqcap S$  and every  $o \in S$  is true, or  $o = \sqcup S$  and there exists  $o \in S$  which is true, or  $o = \neg o'$  and  $o'$  is not true. A constraint  $o$  is false when it is not true.

A unary function  $\alpha obs$  is used to convert a basic constraint to an observability constraint; “ $\bullet$ ” is used to instantiate an observability constraint with a basic constraint and is defined as  $b' \bullet (b \alpha obs) = (b' \wedge b) \alpha obs$  if  $b \in \mathcal{B}$ ;  $b' \bullet \sqcap S = \sqcap \{b' \bullet o \mid o \in S\}$ ;  $b' \bullet \neg o = \neg(b' \bullet o)$ ;  $b' \bullet \sqcup S = \sqcup \{b' \bullet o \mid o \in S\}$ . Operator  $\exists_x o$  is defined as  $\exists_x (b \alpha obs) = (\exists_x b) \alpha obs$ ;  $\exists_x (\sqcap S) = \sqcap \{\exists_x o \mid o \in S\}$ ;  $\exists_x (\sqcup S) = \sqcup \{\exists_x o \mid o \in S\}$ ;  $\exists_x (\neg o) = \neg(\exists_x o)$ .

The set of a decorated tree  $\mathbb{DT}$  is defined as  $\mathbb{DT} = \{(o + G, \varepsilon) \mid o \in \mathcal{O}, G \in \mathbb{G}\} \cup \{(o + G, \tilde{t}) \mid o \in \mathcal{O}, G \in \mathbb{G}, \text{div}(G) \text{ and } \tilde{t} \in \text{Seq}(\mathbb{DT})\}$ .

A Prolog interpreter can visit a node of a decorated tree only if the observability constraint contained in the node is true. If the observability of a node is  $true_o$  it is surely to be visited. The notion of *i-cut-closed* decorated tree is defined in the same way as the *i-cut-closed* tree.

An interpretation gives information for every predicate symbol:

**Definition 3.2** A decorated tree interpretation  $I$  is a map that associates to any predicate symbol  $p$  in a program an element  $I(p)$  of  $\alpha_{\mathbb{DT}}(\mathbb{T})$  which models the behavior of procedure call  $p(\alpha)$  in the program, where  $\alpha$  is a distinguished variable which is not allowed in the syntax of the clauses and  $\alpha_{\mathbb{DT}}$  is a map from  $\text{Seq}^+(\mathbb{T})$  to  $\text{Seq}^+(\mathbb{DT})$ .

The immediate consequence operator  $T_P^{\mathbb{DT}}$  of program  $P$  is defined as

$$T_P^{\mathbb{DT}}(I)(p) = \psi^{p(\alpha)} \exists_y^{\mathbb{DT}} ((true_o + \delta_{y,\alpha}, \varepsilon) \boxtimes^{\mathbb{DT}} \downarrow^{\mathbb{DT}} (\phi^{p(y)} \mathcal{J}_P^{\mathbb{DT}} [G_1] I \boxplus^{\mathbb{DT}} \dots \boxplus^{\mathbb{DT}} \phi^{p(y)} \mathcal{J}_P^{\mathbb{DT}} [G_n] I)),$$

where  $p(x)$ :-  $G_1$  or...or  $G_n$  is the definition of  $p$  in  $P$ ,  $I$  is a decorated tree interpretation and  $\mathcal{J}_P^{\mathbb{DT}}$  is defined as:

$$\begin{aligned} \mathcal{J}_P^{\mathbb{DT}} [c] I &= (true_o + c, \varepsilon) \\ \mathcal{J}_P^{\mathbb{DT}} [p(x)] I &= I(p)[x/\alpha] \\ \mathcal{J}_P^{\mathbb{DT}} [exists\ x.G] I &= \exists_x^{\mathbb{DT}} \mathcal{J}_P^{\mathbb{DT}} [G] I \\ \mathcal{J}_P^{\mathbb{DT}} [cut(G)] I &= !^{\mathbb{DT}} \mathcal{J}_P^{\mathbb{DT}} [G] I \\ \mathcal{J}_P^{\mathbb{DT}} [G_1\ and\ G_2] I &= \mathcal{J}_P^{\mathbb{DT}} [G_1] I \boxtimes^{\mathbb{DT}} \mathcal{J}_P^{\mathbb{DT}} [G_2] I. \end{aligned}$$

We refer to [13] for the definitions of the operators present in  $T_P^{\mathbb{DT}}$  and  $\mathcal{J}_P^{\mathbb{DT}}$ . In this paper, a superscript of an operator symbol is used to indicate which semantics the operator is defined for. For example, symbol  $\boxplus^{\mathbb{DT}}$  denotes the sum operator for DT semantics.

The partial ordering “ $\leq$ ” on  $\alpha_{\mathbb{DT}}(\text{Seq}^+(\mathbb{T}))$  used in the computation process is defined by the following rules.

- 1)  $(o + G, \varepsilon) \leq (o + G, \varepsilon)$ ; 2)  $(o + G, \varepsilon) \leq (o + G, \tilde{t})$ ;
- 3)  $(o + G, \tilde{t}) \leq (o + G, \tilde{t}')$  iff  $\tilde{t} \leq \tilde{t}'$ ;
- 4)  $\tilde{t}_1 \leq \tilde{t}_2$  if for every partition  $\tilde{t}_{11} :: \tilde{t}_{12}$  of  $\tilde{t}_1$ , we can find a partition  $\tilde{t}_{21} :: \tilde{t}_{22}$  of  $\tilde{t}_2$  such that  $\tilde{t}_{11} \leq \tilde{t}_{21}$  and  $\tilde{t}_{12} \leq \neg \sqcup \delta^{\mathbb{DT}}(\tilde{t}_{11}) \square^{\mathbb{DT}} \tilde{t}_{22}$ , where  $\square^{\mathbb{DT}}$  is the instantiation operator that adds further observability constraint to the nodes of a sequence of decorated trees.

This ordering relation can be lifted to a partial ordering on decorated tree interpretation by defining  $I_1 \leq I_2$  if and only if  $I_1(p) \leq I_2(p)$  for every predicate symbol  $p$ . The bottom element of the partial ordering on decorated tree interpretations is denoted by  $I_0^{\mathbb{DT}}$  and is such that  $I_0^{\mathbb{DT}}(p) = (true_o + p(\alpha), \varepsilon)$ . In the completion of the domain of decorated tree interpretations  $\mathcal{F}_P^{\mathbb{DT}}$  is defined as the least fixpoint of  $T_P^{\mathbb{DT}}$ , i.e.  $\mathcal{F}_P^{\mathbb{DT}} = \text{lub}_{\geq 0} (T_P^{\mathbb{DT}}(I_0^{\mathbb{DT}}))^i$ . It is shown that  $\mathcal{F}_P^{\mathbb{DT}}$  is correct decorated tree denotation of program  $P$ .

## 4. Problems in achieving cpa and ccp semantics

In this section we first describe a partial answer (pa) semantics as an easy abstraction of DT semantics. Then we analyze the problems encountered when we are going to achieve cpa or ccp semantics.

Spoto’s DT semantics can be specialized into partial answer (pa) semantics using approaches similar to that for achieving call pattern semantics. Besides convergent, divergent and cut constraint, we need another kind of constraint, i.e. partial answer constraint, to describe partial answers. For example, the partial answer constraint can be of the form  $(o + {}^{pa}b)$  where  $o$  is the observability part and  $b$  is the constraint corresponding to a partial answer. Let  $\mathbb{PA}$  be the set of convergent, divergent, cut and partial answer constraints. We can use

the following abstraction map  $\alpha_{\mathbb{P}\mathbb{A}}: \text{Seq}^+(\mathbb{D}\mathbb{T}) \mapsto \text{Seq}^+(\mathbb{P}\mathbb{A})$

$$\alpha_{\mathbb{P}\mathbb{A}}(o + G, \varepsilon) = \begin{cases} o +^{pa} \text{con}(G) :: o \odot^{\mathbb{P}\mathbb{A}} \text{cutsseq}(G) :: o +^d \text{con}(G) & \text{if } \text{div}(G) \\ o +^{pa} \text{con}(G) :: o \odot^{\mathbb{P}\mathbb{A}} \text{cutsseq}(G) :: o +^c \text{con}(G) & \text{if not } \text{div}(G) \end{cases}$$

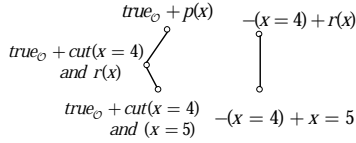
$$\alpha_{\mathbb{P}\mathbb{A}}(o + G, \tilde{t}) = o +^{pa} \text{con}(G) :: o \odot^{\mathbb{P}\mathbb{A}} \text{cutsseq}(G) :: \alpha_{\mathbb{P}\mathbb{A}}(\tilde{t})$$

$$\alpha_{\mathbb{P}\mathbb{A}}(\tilde{t}_1 :: \tilde{t}_2) = \alpha_{\mathbb{P}\mathbb{A}}(\tilde{t}_1) :: \alpha_{\mathbb{P}\mathbb{A}}(\tilde{t}_2),$$

where  $\text{cutsseq}(G)$  is the sequence of cut constraints associated to goal  $G$ , and “ $\odot^{\mathbb{P}\mathbb{A}}$ ” denotes the instantiation operator. Both of them can be defined in much the similar way to those for ca semantics.

**Example 4.1** Given the sequence of decorated trees  $\tilde{t}$  in Fig. 1, its abstraction by  $\alpha_{\mathbb{P}\mathbb{A}}$  is as follows.

$$\begin{aligned} \alpha_{\mathbb{P}\mathbb{A}}(\tilde{t}) &= \text{true}_o +^{pa} \text{true} :: \text{true}_o +^{pa} x = 4 \\ &:: \text{true}_o +^1 x = 4 :: \text{true}_o +^{pa} x = 4 \wedge x = 5 \\ &:: \text{true}_o +^1 x = 4 :: \text{true}_o +^c x = 4 \wedge x = 5 \\ &:: -(x = 4) +^{pa} \text{true} :: -(x = 4) +^{pa} x = 5 \\ &:: -(x = 4) +^c x = 5 \end{aligned}$$



**Fig.1** A sequence  $\tilde{t}$  of decorated trees and its abstraction.

Suppose we want to get a cpa semantics. The problem is how to determine whether a partial answer (or the corresponding pa constraint) is “correct”, i.e. whether it will eventually lead to a successful derivation. In a decorated tree we know that a partial answer  $b$  can correspond to a node  $N$  in the tree, and the partial answer is correct if and only if there is a consistent and observable convergent leaf in the tree rooted at node  $N$ . In our example we know from  $\tilde{t}$  that  $p(x)$  and  $p(4)$  are not correct partial answers of goal  $p(x)$ . But in the pa semantics decorated trees are “flattened” into a sequence of constraints and it’s difficult to determine the essential structure of the original decorated trees only from their abstractions. So in the partial answer semantics we generally cannot tell whether a partial answer constraint is correct or not. In the above example, we cannot tell whether  $\text{true}_o +^{pa} \text{true}$  and  $-(x = 4) +^c x = 5$  belong to the abstraction of the same tree. This means that the abstraction from  $\text{Seq}^+(\mathbb{D}\mathbb{T})$  to  $\text{Seq}^+(\mathbb{P}\mathbb{A})$  is too coarse. We need a finer one for cpa semantics. The similar situation will occur when we are trying to get correct call pattern semantics, i.e. the cp semantics also does not

contain enough information to determine whether a call pattern is “correct”.

Since we are interested in goal-independent semantics, the problems we learn from this example are in two aspects. Firstly, we must have some means to determine from a sequence of constraints the essential structures of the original decorated trees. Secondly, in order to make cpa semantics goal-independent we may have to explicitly specify the conditions under which a partial answer will eventually lead to a successful derivation.

In this paper we argue that the semantics for correct partial answers can be achieved in two steps. The first step deals with the second aspect of problem, where the main task is to abstract DT semantics to an intermediate MDT semantics which contains success condition for the nodes in a decorated tree. By *success condition* we denote the conditions under which a decorated tree node is on at least one successful SLD-derivation. From DT semantics we know the success condition of a decorated tree  $T$  is exactly the convergent condition  $\sqcup \circ^{\text{DT}}(T)$  of  $T$ . The purpose of the first step is to make the final cpa or cp semantics goal-independent. The second step deals with the first aspect of problem and the main task is to achieve cpa semantics or cp semantics as abstractions of MDT semantics. Our solution to the problem is quite natural and the main idea is to make use of the depth-first nature of Prolog search rules. We’ll discuss the second step in detail in other paper. This paper is mainly concerned with the first step.

## 5. Modified Decorated tree (MDT)

In this section we first present a semantic domain  $\text{Seq}(\text{MDT})$ , and then define semantic operators on this domain. By adding success condition to the node of a decorated tree, we get the definition of the set  $\text{MDT}$  of modified decorated trees.

$$\text{MDT} = \{ (o + G + sc, \varepsilon) \mid o, sc \in \mathcal{O}, G \in \mathbb{G} \} \cup \{ (o + G + sc, \tilde{t}) \mid o, sc \in \mathcal{O}, G \in \mathbb{G}, \text{div}(G) \text{ and } \tilde{t} \in \text{Seq}(\text{MDT}) \}$$

where  $sc$  is the success condition of a MDT node.

The abstraction map  $\alpha_{\text{MDT}}$  from  $\text{Seq}^+(\mathbb{D}\mathbb{T})$  to  $\text{Seq}^+(\text{MDT})$  is defined as:

$$\alpha_{\text{MDT}}(o + G, \varepsilon) = \begin{cases} o + G + \text{false}_o & \text{if } \text{div}(G) \\ o + G + o \sqcap \text{con}(G) \propto \text{obs} & \text{if not } \text{div}(G) \end{cases}$$

$$\alpha_{\text{MDT}}(o + G, \tilde{t}) = (o + G + \sqcup \circ^{\text{DT}}(\tilde{t}), \alpha_{\text{MDT}}(\tilde{t}))$$

$$\alpha_{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) = \alpha_{\text{MDT}}(\tilde{t}_1) :: \alpha_{\text{MDT}}(\tilde{t}_2).$$

Note that the main idea here is also the “control compilation”. From the definition of  $\alpha_{\text{MDT}}$  it can be seen that the information for determining whether a goal contained in a node will succeed or not is also compiled.

The following maps are needed to define the abstract counterparts of the operators defined for decorated tree semantics, where  $\circ^{\text{MDT}}$  is used to collect information of convergent leaves and  $\circ^{\text{MDT}}(\tilde{t})$  is true if and only if there is a consistent and observable convergent leaf in  $\tilde{t}$ ,  $k^{\text{MDT}}$  is used to collect cut conditions in the nodes of  $\tilde{t}$ , and  $\delta^{\text{MDT}}$  is used to collect information of divergent leaves in  $\tilde{t}$ .

**Definition 5.1** Given  $\tilde{t}, \tilde{t}_1, \tilde{t}_2 \in \text{Seq}^+(\text{MDT})$ , we define:

$$\begin{aligned} \circ^{\text{MDT}}(o + G + sc, \varepsilon) &= \circ^{\text{MDT}}(o + G + sc, \tilde{t}) = sc \\ \circ^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= \circ^{\text{MDT}}(\tilde{t}_1) \sqcup \circ^{\text{MDT}}(\tilde{t}_2) \\ k^{\text{MDT}}(o + G + sc, \varepsilon) &= o \times (\text{cuts}(G) \times \text{obs}) \\ k^{\text{MDT}}(o + G + sc, \tilde{t}) &= o \times (\text{cuts}(G) \times \text{obs}) \cup \zeta k^{\text{MDT}}(\tilde{t}) \\ k^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= k^{\text{MDT}}(\tilde{t}_1) \cup k^{\text{MDT}}(\tilde{t}_2) \end{aligned}$$

$$\delta^{\text{MDT}}(o + G + sc, \varepsilon) = \begin{cases} \{o \sqcap \text{con}(G) \times \text{obs}\} & \text{if } \text{div}(G) \\ \emptyset & \text{if not } \text{div}(G) \end{cases}$$

$$\begin{aligned} \delta^{\text{MDT}}(o + G + sc, \tilde{t}) &= \delta^{\text{MDT}}(\tilde{t}) \\ \delta^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= \delta^{\text{MDT}}(\tilde{t}_1) \cup \delta^{\text{MDT}}(\tilde{t}_2) \\ \beta^{\text{MDT}}(\tilde{t}) &= k^{\text{MDT}}(\tilde{t}) \cup \delta^{\text{MDT}}(\tilde{t}) \end{aligned}$$

$$\begin{aligned} \xi^{\text{MDT}}((o + G + sc, \varepsilon), T) &= \begin{cases} \emptyset & \text{if } \text{div}(G) \\ \{o \times (\text{con}(G) \bullet \beta^{\text{MDT}}(T))\} & \text{if not } \text{div}(G) \end{cases} \end{aligned}$$

$$\begin{aligned} \xi^{\text{MDT}}((o + G + sc, \tilde{t}), T) &= \zeta \xi^{\text{MDT}}(\tilde{t}, T) \\ \xi^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2, T) &= \xi^{\text{MDT}}(\tilde{t}_1, T) \cup \xi^{\text{MDT}}(\tilde{t}_2, T) \end{aligned}$$

where cut conditions  $\text{cuts}(G)$  of a goal  $G$ , shaking map  $\zeta$  and map  $\times$  are defined in the same way as in [13].

It can be shown that the functions defined in definition 5.1 are correct w.r.t. their counterparts for DT semantics.

**Proposition 5.2** Given  $\tilde{t} \in \text{Seq}^+(\text{DT})$  and  $T \in \text{DT}$ , we have:

- 1)  $\sqcup \circ^{\text{DT}}(\tilde{t}) = \circ^{\text{MDT}}(\alpha_{\text{MDT}}(\tilde{t}))$ ; 2)  $k^{\text{DT}}(\tilde{t}) = k^{\text{MDT}}(\alpha_{\text{MDT}}(\tilde{t}))$ ;
- 3)  $\delta^{\text{DT}}(\tilde{t}) = \delta^{\text{MDT}}(\alpha_{\text{MDT}}(\tilde{t}))$ ; 4)  $\beta^{\text{DT}}(\tilde{t}) = \beta^{\text{MDT}}(\alpha_{\text{MDT}}(\tilde{t}))$ ;
- 5)  $\xi^{\text{DT}}(\tilde{t}, T) = \xi^{\text{MDT}}(\alpha_{\text{MDT}}(\tilde{t}), \alpha_{\text{MDT}}(T))$

Now we are ready to give the definitions of the operators for MDT semantics.

**Definition 5.3** Given a goal  $G$  such that  $\text{div}(G)$ , we define the instantiation operator  $\triangleright^{\text{MDT}}$  for MDT semantics as

$$\begin{aligned} G \triangleright^{\text{MDT}}(o + G' + sc, \varepsilon) &= (\text{con}(G) \bullet o + G \text{ and } G' + \text{con}(G) \bullet sc, \varepsilon) \\ G \triangleright^{\text{MDT}}(o + G' + sc, \tilde{t}) &= (\text{con}(G) \bullet o + G \text{ and } G' + \text{con}(G) \bullet sc, \tau(G) \triangleright^{\text{MDT}}(\tilde{t})) \\ G \triangleright^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= G \triangleright^{\text{MDT}}(\tilde{t}_1) :: G \triangleright^{\text{MDT}}(\tilde{t}_2), \end{aligned}$$

where  $\tau(G)$  is a map that removes all  $\text{cut}()$  or  $\text{cut}_d()$  constructs from goal  $G$  (See definition 5 in [13]).

Let  $o_2 + G_2 + sc_2$  be the root of  $T_2 \in \text{MDT}$ , we define product operator  $\boxtimes^{\text{MDT}}$  as:

$$\begin{aligned} (o_1 + G_1 + sc_1, \varepsilon) \boxtimes^{\text{MDT}} T_2 &= \begin{cases} (o_1 + G_1 \text{ and } G_2 + sc_1, \varepsilon) & \text{if } \text{div}(G_1) \\ o_1 \sqcap^{\text{MDT}}(G_1 \triangleright^{\text{MDT}} T_2) & \text{if not } \text{div}(G_1) \end{cases} \end{aligned}$$

$$\begin{aligned} (o_1 + G_1 + sc_1, \tilde{t}_1) \boxtimes^{\text{MDT}} T_2 &= (o_1 + G_1 \text{ and } G_2 + \circ^{\text{MDT}}(\tilde{t}_1 \boxtimes^{\text{MDT}} T_2), \tilde{t}_1 \boxtimes^{\text{MDT}} T_2) \\ (\tilde{t}_1 :: \tilde{t}_2) \boxtimes^{\text{MDT}} T_2 &= \tilde{t}_1 \boxtimes^{\text{MDT}} T_2 :: - \sqcup \xi^{\text{MDT}}(\tilde{t}_1, T_2) \sqcap^{\text{MDT}}(\tilde{t}_2 \boxtimes^{\text{MDT}} T_2), \end{aligned}$$

where

$$\begin{aligned} o \sqcap^{\text{MDT}}(o_1 + G + sc, \varepsilon) &= (o \sqcap o_1 + G + o \sqcap sc, \varepsilon) \\ o \sqcap^{\text{MDT}}(o_1 + G + sc, \tilde{t}) &= (o \sqcap o_1 + G + o \sqcap sc, o \sqcap^{\text{MDT}} \tilde{t}) \\ o \sqcap^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= (o \sqcap^{\text{MDT}} \tilde{t}_1 :: o \sqcap^{\text{MDT}} \tilde{t}_2). \end{aligned}$$

$$\begin{aligned} \exists_x^{\text{MDT}}(o + G + sc, \varepsilon) &= (\exists_x o + \text{exists } x.G + \exists_x sc, \varepsilon) \\ \exists_x^{\text{MDT}}(o + G + sc, \tilde{t}) &= (\exists_x o + \text{exists } x.G + \exists_x sc, \exists_x^{\text{MDT}}(\tilde{t})) \\ \exists_x^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= \exists_x^{\text{MDT}}(\tilde{t}_1) :: \exists_x^{\text{MDT}}(\tilde{t}_2) \end{aligned}$$

We define cut operator  $!^{\text{MDT}}$  and uncut operator  $\downarrow^{\text{MDT}}$  as:

$$\begin{aligned} !^{\text{MDT}}(o + G + sc, \varepsilon) &= (o + \text{cut}(G) + sc, \varepsilon) \\ !^{\text{MDT}}(o + G + sc, \tilde{t}) &= o + \text{cut}(G) + \circ^{\text{MDT}}(\tilde{t}), !^{\text{MDT}}(\tilde{t}) \\ !^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= !^{\text{MDT}}(\tilde{t}_1) :: - \circ^{\text{MDT}}(\tilde{t}_1) \sqcap^{\text{MDT}} !^{\text{MDT}}(\tilde{t}_2) \\ \downarrow^{\text{MDT}}(\tilde{t}) &= \downarrow_0^{\text{MDT}}(\tilde{t}) \\ \downarrow_i^{\text{MDT}}(o + G + sc, \varepsilon) &= (o + \downarrow_i G + sc, \varepsilon) \\ \downarrow_i^{\text{MDT}}(o + G + sc, \tilde{t}) &= (o + \downarrow_i G + sc, \downarrow_{i+1}^{\text{MDT}} \tilde{t}) \\ \downarrow_i^{\text{MDT}}(\tilde{t}_1 :: \tilde{t}_2) &= \downarrow_i^{\text{MDT}} \tilde{t}_1 :: \downarrow_i^{\text{MDT}} \tilde{t}_2, \end{aligned}$$

where  $\downarrow_i$  is defined in the same way as in [13].

Expansion  $\phi_{\text{MDT}}^G$ , root swapping  $\psi_{\text{MDT}}^G$ , sum  $\boxplus^{\text{MDT}}$ , and substitution operator  $\square$  are defined, respectively as:

$$\begin{aligned} \phi_{\text{MDT}}^G(T) &= (\text{true}_o + G + \circ^{\text{MDT}}(T), T) \\ \psi_{\text{MDT}}^G(o + G' + sc, \varepsilon) &= (o + G + sc, \varepsilon) \\ \psi_{\text{MDT}}^G(o + G' + sc, \tilde{t}) &= (o + G + sc, \tilde{t}) \\ (o + G + c_1, \tilde{t}_1) \boxplus^{\text{MDT}}(o + G + c_2, \tilde{t}_2) &= (o + G + c_1 \\ &\quad \sqcup (- \sqcup \beta^{\text{MDT}}(\tilde{t}_1) \sqcap c_2), \tilde{t}_1 :: - \sqcup \beta^{\text{MDT}}(\tilde{t}_1) \sqcap^{\text{MDT}} \tilde{t}_2) \\ (\text{true}_o + p(\alpha) + sc, \exists_x^{\text{MDT}}((\text{true}_o + \delta_{x,\alpha} + \delta_{x,\alpha} \times \text{obs}, \varepsilon) \\ &\quad \boxtimes^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t}))) [x / \alpha] \\ &= (\text{true}_o + p(x) + sc[\text{true} / \delta_{x,\alpha}], \alpha_{\text{MDT}}(\tilde{t})) \\ (\text{true}_o + p(\alpha) + sc, \exists_y^{\text{MDT}}((\text{true}_o + \delta_{y,\alpha} + \delta_{y,\alpha} \times \text{obs}, \varepsilon) \\ &\quad \boxtimes^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t}))) [x / \alpha] \\ &= (\text{true}_o + p(x) + sc[\delta_{y,x} / \delta_{y,\alpha}], \\ &\quad \exists_y^{\text{MDT}}((\text{true}_o + \delta_{y,x} + \delta_{y,x} \times \text{obs}, \varepsilon) \boxtimes^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t}))) \end{aligned}$$

where  $sc[b_1/b_2]$  denotes the operation that substitutes every occurrence of  $b_2$  in  $sc$  with  $b_1$ . Note that in the operation  $T[x/\alpha]$  we substitute  $\delta_{x,\alpha}$  with  $\text{true}$  in the first case and  $\delta_{y,\alpha}$  with  $\delta_{y,x}$  in the second case, because in our semantics every occurrence of  $\alpha$  is in the form of  $\delta_{x,\alpha}$  or  $\delta_{y,\alpha}$ , as will be seen later in the definition of  $T_p^{\text{MDT}}$ .

All the operators defined in definition 5.3 are correct with respect to their counterparts defined in [13] for DT semantics.

**Proposition 5.4**

- (1) Given  $\tilde{t} \in \text{Seq}^+(\mathbb{DT})$  and  $G \in \mathbb{G}$  such that  $\text{not div}(G)$ , we have :  
 $\alpha_{\text{MDT}}(G \triangleright^{\text{DT}} \tilde{t}) = G \triangleright^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t})$ .
- (2) Given  $\tilde{t} \in \text{Seq}^+(\mathbb{DT})$  and  $o \in \mathcal{O}$ , we have:  
 $\alpha_{\text{MDT}}(o \sqsupset^{\text{DT}} \tilde{t}) = o \sqsupset^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t})$ .
- (3) Given  $\tilde{t} \in \text{Seq}^+(\mathbb{DT})$  and  $T \in \mathbb{DT}$ , we have:  
 $\alpha_{\text{MDT}}(\tilde{t} \boxtimes^{\text{DT}} T) = \alpha_{\text{MDT}}(\tilde{t}) \boxtimes^{\text{MDT}} \alpha_{\text{MDT}}(T)$
- (4) Given  $\tilde{t}_1, \tilde{t}_2 \in \text{Seq}^+(\mathbb{DT})$ ,  $o \in \mathcal{O}$  and  $G \in \mathbb{G}$ , we have:  
 $\alpha_{\text{MDT}}((o + G, \tilde{t}_1) \boxplus^{\text{DT}} (o + G, \tilde{t}_2)) = \alpha_{\text{MDT}}(o + G, \tilde{t}_1) \boxplus^{\text{MDT}} \alpha_{\text{MDT}}(o + G, \tilde{t}_2)$ .
- (5) Given  $\tilde{t} \in \text{Seq}^+(\mathbb{DT})$ , we have:  
 $\alpha_{\text{MDT}}(\text{!}^{\text{DT}} \tilde{t}) = \text{!}^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t})$ .
- (6) Given  $\tilde{t} \in \text{Seq}^+(\mathbb{DT})$ , we have:  
 $\alpha_{\text{MDT}}(\text{?}_x^{\text{DT}} \tilde{t}) = \text{?}_x^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t})$ .
- (7) Given  $\tilde{t} \in \text{Seq}^+(\mathbb{DT})$  and  $i \geq 0$ , we have:  
 $\alpha_{\text{MDT}}(\downarrow_i^{\text{DT}} \tilde{t}) = \downarrow_i^{\text{MDT}} \alpha_{\text{MDT}}(\tilde{t})$ .
- (8) Given  $T \in \mathbb{DT}$ , we have:  
 $\alpha_{\text{MDT}}(T[x/\alpha]) = \alpha_{\text{MDT}}(T)[x/\alpha]$ .
- (9) Given  $T \in \mathbb{DT}$ , we have:  
 $\alpha_{\text{MDT}}(\phi_{\text{DT}}^G(T)) = \phi_{\text{MDT}}^G(\alpha_{\text{MDT}}(T))$ .
- (10) Given  $T \in \mathbb{DT}$  whose height is at least 2 and  $G \in \mathbb{G}$ , we have:  $\alpha_{\text{MDT}}(\psi_{\text{DT}}^G(T)) = \psi_{\text{MDT}}^G(\alpha_{\text{MDT}}(T))$ .

**Definition 5.5** A *MDT interpretation*  $I$  is a map which associates to every predicate symbol  $p$  in a program an element  $I(p)$  of  $\alpha_{\text{MDT}}(\alpha_{\text{DT}}(\text{Seq}^+(\mathbb{T})))$  which models the behavior of procedure call  $p(\alpha)$  in the program, where  $\alpha$  is a variable not allowed in all the programs.

With the operators defined in definition 5.3 we can give the abstract immediate consequence operator  $T_P^{\text{MDT}}$ .

$$T_P^{\text{MDT}}(I)(p) = \psi^{p(\alpha)} \exists_y^{\text{MDT}} ((\text{true}_o + \delta_{y,\alpha} + \delta_{y,\alpha'}, \varepsilon) \boxtimes^{\text{MDT}} \downarrow^{\text{MDT}} (\phi^{p(y)} \mathcal{J}_P^{\text{MDT}} [G_1] I \boxplus^{\text{MDT}} \dots \boxplus^{\text{MDT}} \phi^{p(y')} \mathcal{J}_P^{\text{MDT}} [G_n] I))$$

where  $I$  is a MDT interpretation and  $\mathcal{J}_P^{\text{MDT}}$  is defined as:

$$\begin{aligned} \mathcal{J}_P^{\text{MDT}} [c] I &= (\text{true}_o + c + c, \varepsilon) \\ \mathcal{J}_P^{\text{MDT}} [p(x)] I &= I(p)[x/\alpha] \\ \mathcal{J}_P^{\text{MDT}} [\text{exists } x.G] I &= \exists_x^{\text{MDT}} \mathcal{J}_P^{\text{MDT}} [G] I \\ \mathcal{J}_P^{\text{MDT}} [\text{cut}(G)] I &= \text{!}^{\text{MDT}} \mathcal{J}_P^{\text{MDT}} [G] I \\ \mathcal{J}_P^{\text{MDT}} [G_1 \text{ and } G_2] I &= \mathcal{J}_P^{\text{MDT}} [G_1] I \boxtimes^{\text{MDT}} \mathcal{J}_P^{\text{MDT}} [G_2] I \end{aligned}$$

Let  $\mathcal{J}_P^{\text{DT}}$  and  $T_P^{\text{DT}}$  be the counterparts of  $\mathcal{J}_P^{\text{MDT}}$  and  $T_P^{\text{MDT}}$  in DT semantics, respectively. By proposition 5.4 we have the following theorem:

**Theorem 5.6** Let  $I$  be a DT interpretation and  $G$  be a goal. We have:

- 1)  $\alpha_{\text{MDT}}(\mathcal{J}_P^{\text{DT}} [G] I) = \mathcal{J}_P^{\text{MDT}} [G] \alpha_{\text{MDT}}(I)$
- 2)  $\alpha_{\text{MDT}}(T_P^{\text{DT}}(I)) = T_P^{\text{MDT}}(\alpha_{\text{MDT}}(I))$ .

## 6. MDT semantics and its correctness

In this section we define an ordering  $\sqsubseteq$  on  $\text{Seq}^+(\text{MDT})$ , which turns out to be a partial ordering on  $\alpha_{\text{MDT}}(\alpha_{\text{DT}}(\text{Seq}^+(\mathbb{T})))$ . This ordering is lifted to a partial ordering on the domain of MDT interpretations. Then the least fixpoint  $\mathcal{F}_P^{\text{MDT}}$  of  $T_P^{\text{MDT}}$  in the completion of the domain is shown to be a correct abstraction of the decorated tree denotation  $\mathcal{F}_P^{\text{DT}}$  of program  $P$ . We first give the relation  $\sqsubseteq$  and then prove that  $\sqsubseteq$  is really the relation we need.

**Definition 6.1** On  $\text{Seq}^+(\text{MDT})$  we define the relation  $\sqsubseteq$  as follows:

- 1)  $(o + G + sc, \varepsilon) \sqsubseteq (o + G + sc, \varepsilon)$
- 2)  $(o + G + sc_1, \varepsilon) \sqsubseteq (o + G + sc_2, \tilde{t})$  iff  $sc_1 \leq sc_2$
- 3)  $(o + G + sc_1, \tilde{t}_1) \sqsubseteq (o + G + sc_2, \tilde{t}_2)$  iff  $\tilde{t}_1 \sqsubseteq \tilde{t}_2$  and  $sc_1 \leq sc_2$
- 4)  $\tilde{t}_1 \sqsubseteq \tilde{t}_2$  if for every partition  $\tilde{t}_{11} :: \tilde{t}_{12}$  of  $\tilde{t}_1$ , we can find a partition  $\tilde{t}_{21} :: \tilde{t}_{22}$  of  $\tilde{t}_2$  such that  $\tilde{t}_{11} \sqsubseteq \tilde{t}_{21}$  and  $\tilde{t}_{12} \sqsubseteq - \sqcup \delta^{\text{MDT}}(\tilde{t}_{11}) \sqcup^{\text{MDT}} \tilde{t}_{22}$ .

This ordering relation can be lifted to the domain of MDT interpretations by defining  $I_1 \sqsubseteq I_2$  iff  $I_1(p) \sqsubseteq I_2(p)$  for every predicate  $p$  in a program. We show that  $\sqsubseteq$  is indeed a partial ordering relation on  $\alpha_{\text{MDT}}(\alpha_{\text{DT}}(\text{Seq}^+(\mathbb{T})))$ .

**Theorem 6.2**  $\sqsubseteq$  is a partial order relation on  $\alpha_{\text{MDT}}(\alpha_{\text{DT}}(\text{Seq}^+(\mathbb{T})))$ .

The bottom element of this partial ordering on MDT interpretations is denoted by  $I_0^{\text{MDT}}$  and is such that  $I_0^{\text{MDT}}(p) = (\text{true}_o + p(\alpha) + \text{false}_o, \varepsilon)$  for every predicate symbol  $p$ . It can be shown that  $I_0^{\text{MDT}}$  is strict.

$$\begin{aligned} \alpha_{\text{MDT}}(I_0^{\text{DT}}(p)) &= \alpha_{\text{MDT}}(\text{true}_o + p(\alpha), \varepsilon) \\ &= (\text{true}_o + p(\alpha) + \text{false}_o, \varepsilon) = I_0^{\text{MDT}}(p) \end{aligned}$$

The following theorem shows that  $\alpha_{\text{MDT}}$  is a monotonic map from  $\leq$  to  $\sqsubseteq$  and  $T_P^{\text{MDT}}$  is monotonic w.r.t.  $\sqsubseteq$ . By the monotonicity of  $\sqcup \circ^{\text{DT}}$  and induction on  $\# \tilde{t}_1$  we have the following theorem.

**Theorem 6.3** Given  $\tilde{t}_1, \tilde{t}_2 \in \alpha_{\text{DT}}(\text{Seq}^+(\mathbb{T}))$  such that  $\tilde{t}_1 \leq \tilde{t}_2$ , we have:  $\alpha_{\text{MDT}}(\tilde{t}_1) \sqsubseteq \alpha_{\text{MDT}}(\tilde{t}_2)$ .

Note that as a simple consequence of definition 6.1, the converse of Theorem 6.3 also holds, i.e.,

$\alpha_{\text{MDT}}(\tilde{t}_1) \sqsubseteq \alpha_{\text{MDT}}(\tilde{t}_2)$  implies  $\tilde{t}_1 \leq \tilde{t}_2$ . This allows us to conclude that  $T_P^{\text{MDT}}$  is monotonic on  $\alpha_{\text{MDT}}(\alpha_{\text{DT}}(\text{Seq}^+(\mathbb{T})))$ .

**Theorem 6.4** Given  $\tilde{t}_1, \tilde{t}_2 \in \alpha_{\text{MDT}}(\alpha_{\text{DT}}(\text{Seq}^+(\mathbb{T})))$  such that  $\tilde{t}_1 \sqsubseteq \tilde{t}_2$ , we have  $T_P^{\text{MDT}}(\tilde{t}_1) \sqsubseteq T_P^{\text{MDT}}(\tilde{t}_2)$ .

Then in the completion of the domain of MDT interpretations we can define  $\mathcal{F}_P^{\text{MDT}} = \text{lub}_{i \geq 0} (T_P^{\text{MDT}})^i$  (Here  $(T_P^{\text{MDT}})^i$  is a shorthand for  $(T_P^{\text{MDT}}(I_0^{\text{MDT}}))^i$ ). In section 5 we have shown that  $T_P^{\text{MDT}}$  is correct w.r.t.  $T_P^{\text{DT}}$ . Then from the above theorems and proposition 2.3 and 2.4, we conclude that

**Theorem 6.5** Given program  $P$ , the following equalities hold.

$$\begin{aligned} \alpha_{\text{MDT}}(\mathcal{F}_P^{\text{DT}}) &= \alpha_{\text{MDT}}(\text{lfp}(T_P^{\text{DT}})) = \text{lfp}(T_P^{\text{MDT}}) \\ &= \text{lub}_i (T_P^{\text{MDT}})^i = \mathcal{F}_P^{\text{MDT}} \end{aligned}$$

By Theorem 6.5,  $\mathcal{F}_P^{\text{MDT}}$  is a correct denotation of program  $P$ . Theorem 5.6 and 6.5, together with the definition of  $\alpha_{\text{MDT}}$  allow us to draw the following conclusion.

**Theorem 6.6** Given a program  $P$  and a goal  $G$ , the set of correct partial answers of  $G$  is given by  $\{con(G)|(o+G+sc)\}$  is a node of  $\mathcal{J}_P^{\text{MDT}}[G]\mathcal{F}_P^{\text{MDT}}$  whose success condition  $sc$  is true }

Similarly we have a theorem for correct call patterns.

**Theorem 6.7** Given a program  $P$  and a goal  $G$ , the set of correct call patterns of  $G$  is given by  $\{con(G),p|(o+G+sc)\}$  is a node of  $\mathcal{J}_P^{\text{MDT}}[G]\mathcal{F}_P^{\text{MDT}}$  whose success condition  $sc$  is true and  $p$  is the leftmost procedure call present in  $G$ .

## 7. Conclusion

This paper proposes a two-step method for achieving semantics for correct partial answers and correct call patterns from Spoto's decorated tree semantics. An intermediate MDT semantics is presented, which contains information related to successful derivations and therefore allows for further abstraction to obtain cpa and ccp semantics. With the theory of abstract interpretation, this semantics is shown to be correct with respect to decorated tree semantics. Theorems in this paper show that MDT semantics is sufficient for describing correct partial answers and correct call patterns.

Future work includes the derivation of computable abstract cpa and ccp semantics from the semantics achieved by our method and their use in the analysis of non-trivial applications.

## 8. Acknowledgment

Supported by National Natural Science Foundation of China (No.60563005).

## 9. References

- [1] R. Barbuti, M. Codish, R. Giacobazzi, G. Levi. Modelling Prolog Control, *Journal of Logic and Computation*, 3: 579-603, 1993.
- [2] A. Bossi, M. Bugliesi, M. Fabris, Fixpoint Semantics for Prolog, in: D.S. Warren (Ed.), *Proceedings of the Tenth International Conference on Logic Programming*, MIT Press, Cambridge, MA, 1993, pp. 374-389.
- [3] A. Bossi, M. Gabbrielli, G. Levi, M. Martelli. The S-Semantics Approach: Theory and Applications, *Journal of Logic Programming*, 19-20: 149-197, 1994.
- [4] P. Cousot, R. Cousot. Abstract Interpretation and Applications to Logic Programs, *Journal of Logic Programming* 13 (23): 103-179, 1992.
- [5] P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation, *Theoretical Computer Science*, 277(1-2): 47-103, 2002.
- [6] M. Gabbrielli, G. Levi, M.C. Meo. Observable Behaviors and Equivalences of Logic Programs, *Information and Computation*, 122(1): 1-29, 1995.
- [7] L. Henkin, J.D. Monk, A. Tarski. *Cylindric Algebras. Parts I and II*, North-Holland, Amsterdam, 1971.
- [8] J. Jaffar, M.J. Maher. Constraint Logic programming: a Survey, *Journal of Logic Programming* 19-20: 503-581, 1994.
- [9] B. Le Charlier, S. Rossi, P. Van Hentenryck. An Abstract Interpretation Framework Which Accurately Handles Prolog Search Rule and the Cut, in: M. Bruynooghe (Ed.), *Proceedings of the 1994 International Symposium on Logic Programming*, MIT Press, Cambridge, MA, 1994, pp. 157-171.
- [10] B. Le Charlier, S. Rossi, P. Van Hentenryck. Sequence-based Abstract Interpretation of Prolog, *Theory and Practice of Logic Programming*, v2, n1, 25-84, 2002.
- [11] G. Levi, D. Micciancio. Analysis of Pure Prolog Programs, in: M.I. Sessa (Ed.), *Proceedings GULP-PRODE '95*, 1995, pp. 521-532.
- [12] G. Levi, F. Spoto. Accurate Analysis of Prolog with Cut, in: P. Lucio, M. Martelli, M. Navarro (Eds.), *Proceeding APPIA-GULP-PRODE'96*, 1996, pp. 481-492.
- [13] F. Spoto. Operational and Goal-independent Denotational Semantics for Prolog with cut. *The Journal of Logic Programming*, 42: 1-46, 2000.
- [14] F. Spoto, Giorgio Levi. Abstract Interpretation of Prolog Programs, in: A.M. Haeberer (Ed.), *Proceedings of the Seventh International Conference on Algebraic Methodology and Software Technology, AMAST'98*, Lecture Notes in Computer Science, vol. 1548, Amazonia, Manaus, Brazil, January, Springer, Berlin, 1999, pp. 455-470.
- [15] A. Tarski. A lattice Theoretical Fixpoint Theorem and Its Applications. *Pacific Journal of Mathematics*, 5:285-310, 1955.